Sparse Binary Compression: Towards Distributed Deep Learning with minimal Communication

Felix Sattler
Dept. of Video Coding and Analytics
Fraunhofer HHI
Berlin, Germany
felix.sattler@hhi.fraunhofer.de

Simon Wiedemann
Dept. of Video Coding and Analytics
Fraunhofer HHI
Berlin, Germany
simon.wiedemann@hhi.fraunhofer.de

Klaus-Robert Müller
Machine Learning Group
TU Berlin
Berlin, Germany
klaus-robert.mueller@tu-berlin.de

Wojciech Samek
Dept. of Video Coding and Analytics
Fraunhofer HHI
Berlin, Germany
wojciech.samek@hhi.fraunhofer.de

***Abstract*—Currently, progressively larger deep neural networks are trained on ever growing data corpora. In result, distributed training schemes are becoming increasingly relevant. A major issue in distributed training is the limited communication bandwidth between contributing nodes or prohibitive communication cost in general. To mitigate this problem we propose Sparse Binary Compression (SBC), a compression framework that allows for a drastic reduction of communication cost for distributed training. SBC combines existing techniques of communication delay and gradient sparsification with a novel binarization method and optimal weight update encoding to push compression gains to new limits. By doing so, our method also allows us to smoothly trade-off gradient sparsity and temporal sparsity to adapt to the requirements of the learning task. Our experiments show, that SBC can reduce the upstream communication on a variety of convolutional and recurrent neural network architectures by more than *four* orders of magnitude without significantly harming the convergence speed in terms of forward-backward passes. For instance, we can train ResNet50 on ImageNet in the same number of iterations to the baseline accuracy, using $\times 3531$ less bits or train it to a 1% lower accuracy using $\times 37208$ less bits. In the latter case, the total upstream communication required is cut from 125 *terabytes* to 3.35 gigabytes for every participating client.**

***Index Terms*—distributed optimization, deep learning, communication, efficiency**

I. Introduction

Distributed Stochastic Gradient Descent (DSGD) is a training setting, in which a number of clients jointly trains a deep learning model using stochastic gradient descent [1] [2] [3]. Every client holds an individual subset of the training data, used to improve the current master model. The improvement is obtained by investing computational resources to perform iterations of stochastic gradient descent (SGD). This local training produces a weight update $\Delta\mathcal{W}$ in every participating client, which in regular or irregular intervals ("communication rounds") is exchanged to produce a new master model. This exchange of weight updates can be performed indirectly via a centralized server or directly in an all-reduce operation. In both cases, all clients share the same master model after every communication round (see Fig. 1). In vanilla DSGD the clients have to communicate a full gradient update during every iteration. Every such update is of the same size as the full model, which can be in the range of gigabytes for modern architectures with millions of parameters [4] [5]. Over the course of multiple hundred thousands of training iterations on big datasets the total communication for every client can easily grow to more than a *petabyte*. Consequently, if communication bandwidth is limited, or communication is costly, distributed deep learning can become unproductive or even unfeasible. DSGD is a very popular training setting with many applications. On one end of the spectrum, DSGD can be used to greatly reduce the training time of large-scale deep learning models by introducing device-level data parallelism [6] [7] [8] [9], making use of the fact that the computation of a mini-batch gradient is perfectly parallelizable. In this setting, the clients are usually embodied by hardwired high-performance computation units (i.e. GPUs in a cluster) and every client performs one iteration of SGD per communication round. Since communication is high-frequent in this setting, bandwidth can be a significant bottleneck. On the other end of the spectrum DSGD can also be used to enable privacy-preserving deep learning [10] [11]. Since the clients only ever share weight updates, DSGD makes it possible to train a model from the combined data of all clients without any individual client having to reveal their local training data to a centralized server. In this setting the clients typically are embedded or mobile devices with low network bandwidth, intermittent network connections, and an expensive mobile data plan.

In both scenarios, the communication cost between the individual training nodes is a limiting factor for the performance of the whole learning system. For the synchronous distributed training scheme described above, the total amount of bits

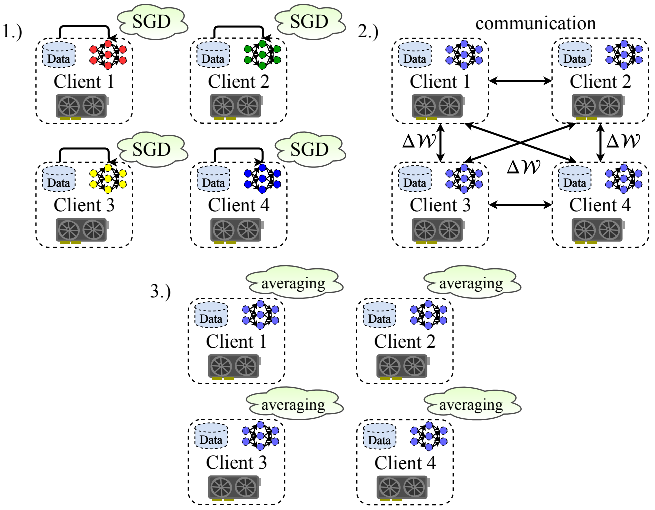

Fig. 1: One communication round of communication-efficient DSGD: 1.) Clients compute a weight-update based on their local data. 2.) Clients exchange weight-updates. 3.) Clients aggregate the weight-updates. Exchange and aggregation of weight-updates can also be performed via a parameter server as described in Algorithm 1.

communicated by every client during training is given by

$$\mathrm{b}_{total} \in \mathcal{O}(\underbrace{N_{iter} \times f}_{\text{\# communication rounds}} \times \underbrace{|\Delta\mathcal{W}_{\neq 0}| \times (\bar{\mathrm{b}}_{pos} + \bar{\mathrm{b}}_{val}))}_{\text{\# bits per communication}} \tag{1}$$

where N_{iter} is the total number of training iterations (forward-backward passes) every client performs, f is the communication frequency, $|\mathcal{W}_{\neq 0}|$ is the sparsity of the weight update and $\bar{\mathrm{b}}_{pos}$, $\bar{\mathrm{b}}_{val}$ respectively are the average number of bits required to communicate the position and the value of the non-zero elements (if $\mathcal{W}$ is dense, the positions of all weights are predetermined and no position bits are required).

Substantial research has gone into the effort of reducing the amount of communication necessary between the clients via lossy compression schemes. Using the systematic of (1), we can organize these prior approaches into three different groups:

Sparsification methods restrict weight updates to modifying only a small subset of the parameters, thus reducing $|\Delta\mathcal{W}_{\neq 0}|$. Strom [12] presents an approach (later modified by Tsuzuki et al. [13]) in which only gradients with a magnitude greater than a certain predefined threshold are sent to the server. All other gradients are aggregated into a residual. This method achieves compression rates of up to 3 orders of magnitude on an acoustic modeling task. In practice however, it is hard to choose appropriate values for the threshold, as it may vary a lot for different architectures and even different layers. Instead of using a fixed threshold to decide what gradient entries to send, Aji et al. [14] use a fixed sparsity rate. They only communicate the fraction p entries of the gradient with the biggest magnitude, while also collecting all other gradients in a residual. At a sparsity rate of $p = 0.001$ their method slightly degrades the convergence speed and final accuracy of the trained model. Lin et al. [15] present modifications to the work of Aji et al. which close this performance gap. These modifications include using a curriculum to slowly increase the amount of sparsity in the first couple communication rounds and applying momentum factor masking to overcome the problem of gradient staleness. Their method achieves compression rates ranging from ×270 to ×600 on different architectures, without slowdown in convergence speed. The convergence of sparsified SGD has been established under differently strong assumptions for both strongly convex and non-convex objective functions [16] [17].

Communication delay methods try to reduce the communication frequency f. McMahan et al. [11] propose Federated Averaging to reduce the cumulative communication. In Federated Averaging, instead of communicating after every iteration, every client performs multiple iterations of SGD to compute a weight update. The authors observe that this delay of communication does not significantly harm the convergence speed in terms of local iterations and report a reduction in the number of necessary communication rounds by a factor of ×10 - ×100 on different convolutional and recurrent neural network architectures. In a follow-up work Konevcny et al. [18] combine this communication delay with random sparsification and probabilistic quantization. They restrict the clients to learn random sparse weight updates or force random sparsity on them afterwards ("structured" vs "sketched" updates) and combine this sparsification with probabilistic quantization. While their method also combines communication delay with (random) sparsification and quantization, and achieves good compression gains for one particular CNN and LSTM model, it also causes a major drop in convergence speed and final accuracy.

Dense quantization methods try to reduce the amount of value bits $\bar{\mathrm{b}}_{val}$. Different quantization methods have been proposed that reduce the bit-width of the gradients to ternary [19], binary [20] [21] or arbitrary [22] bitwidths. Like the sparsification methods discussed above these methods also benefit from error accumulation [23]. While these methods are theoretically well-founded and often come with convergence guarantees, they are also limited to a maximum compression rate of ×32, compared to the regular 32-bit encoding.

II. On the Accumulation of Gradient Information

Communication delay and accumulating sparsification methods as described above already achieve impressive compression rates, however the phenomenon underlying their successes is still only poorly understood. We present a new information-theoretic perspective that is based on the observation that both of these approaches achieve compression by accumulating gradient information locally before sending it to the server. In the case of communication delay all gradients are accumulated uniformly for a fixed amount of iterations, while in the case of sparsification methods they are accumulated non-uniformly until they exceed some fixed or adaptive threshold. In both cases the rate of compression is proportional to the number of steps that the updates are being delayed on average.

Consider now the optimization path $\Delta\mathcal{W}_1, .., \Delta\mathcal{W}_T$ taken by SGD on the loss-surface between some initialization point $\mathcal{W}_0$ and the model $\mathcal{W}_T = \mathcal{W}_0 + \sum_{t=1}^{T} \Delta\mathcal{W}_t$ trained for T iterations. Following this path, we can model the changes occurring to any individual weight in the network w as a noisy stochastic process via

$$\Delta w^t = s^t + n^t, \; t = 1, .., T \tag{2}$$

where s^t denotes the deterministic signal (i.e. the true direction of the minimum), while n^t denotes the noise, induced by mini-batch sampling in SGD ("batch noise") and the stochasticity of the learning process itself ("optimization noise", see Fig. 2 for an illustration). Motivated by the central limit theorem and empirical investigations by [24] we can make the simplifying assumption (a) that this noise n^t is normally distributed at every time-step $n^t \sim \mathcal{N}(0, \sigma^2)$ with the variance being constant in time $\mathbb{V}(n^t) = \sigma^2$ for all $t = 1, .., T$. Since the optimization process has the tendency to damp noise as investigated for instance in [25] it is also reasonable to assume (b) that the noise is (negatively) self-correlated. The noise process is then given by $n^1 = N^1$, $n^t = \alpha n^{t-1} + N^t$, with N^t normally distributed and all N^t uncorrelated, $\alpha \in (-1, 0)$. Given these assumptions it is straight-forward to bound the variance of the accumulated parameter updates.

Proposition 1. *Under assumptions (a) and (b), the variance of the accumulated noise can be bounded by*

$$\mathbb{V}(\sum_{t=1}^{T} n^t) \leq \sigma^2(T(1+\alpha)+1). \tag{3}$$

Proof. Since

$$\begin{aligned} n^t = \alpha n^{t-1} + N^t &= \alpha(\alpha n^{t-2} + N^{t-1}) + N^t \\ &= \alpha^2 n^{t-2} + \alpha N^{t-1} + N^t \\ &= \alpha^\tau n^{t-\tau} + \sum_{i=0}^{\tau-1} \alpha^i N^{t-i} \end{aligned} \tag{4}$$

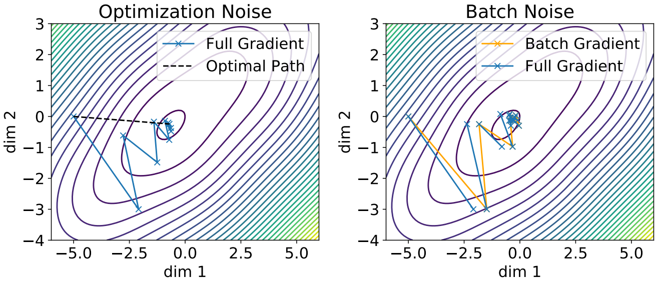

Fig. 2: Sources of noise in SGD (illustration): Left: Optimization noise, caused by Gradient Descent overshooting. Bouncing between the walls of the ravine results in negatively correlated noise. Right: Batch noise, caused by the batch loss being only a noisy approximation of the full empirical loss.

it holds that

$$\begin{aligned} \mathrm{cov}(n^{t-\tau}, n^t) &= \mathrm{cov}(n^{t-\tau}, \alpha^\tau n^{t-\tau} + \sum_{i=0}^{\tau-1} \alpha^i N^{t-i}) \\ &= \alpha^\tau \sigma^2 + \sum_{i=0}^{\tau-1} \alpha^i \underbrace{\mathrm{cov}(n^{t-\tau}, N^{t-i})}_{=0} = \alpha^\tau \sigma^2 \end{aligned} \tag{5}$$

With equation (5) it follows that

$$\mathbb{V}(\sum_{t=1}^{T} n^t) = \sum_{t_1=1}^{T} \sum_{t_2=1}^{T} \mathrm{cov}(n^{t_1}, n^{t_2}) \tag{6}$$

$$= \underbrace{\sum_{t=1}^{T} \mathrm{cov}(n^t, n^t)}_{T\sigma^2} + 2 \underbrace{\sum_{t=1}^{T-1} \mathrm{cov}(n^t, n^{t+1})}_{\alpha(T-1)\sigma^2} + \tag{7}$$

$$2 \underbrace{\sum_{t=1}^{T-2} \mathrm{cov}(n^t, n^{t+2})}_{\alpha^2(T-2)\sigma^2} + .. + 2 \underbrace{\mathrm{cov}(n^1, n^T)}_{\alpha^{T-1}(1)\sigma^2} \tag{8}$$

For negatively correlated noise $\alpha \in (-1, 0)$ we can bound this term by

$$\mathbb{V}(\sum_{t=1}^{T} n^t) = \sigma^2(T + 2 \sum_{\tau=1}^{T-1} \alpha^\tau (T - \tau)) \tag{9}$$

$$= \sigma^2(T + 2 \frac{\alpha^{T+1} - \alpha^2 T + \alpha T - \alpha}{(\alpha - 1)^2}) \tag{10}$$

$$= \sigma^2(T + 2 \underbrace{\frac{(\alpha - \alpha^2)}{(\alpha - 1)^2}}_{\leq \frac{1}{2}\alpha} T + 2 \underbrace{\frac{\alpha^{T+1} - \alpha}{(\alpha - 1)^2}}_{\leq \frac{1}{2}}) \tag{11}$$

$$\leq \sigma^2(T(1+\alpha)+1) \tag{12}$$

□

Theorem 1 directly leads us to a lower bound on the signal-to-noise ratio of the accumulated weight-updates:

Corollary 1. *Under assumptions (a) and (b), accumulation increases the signal-to-noise ratio from $\bar{s}/\sigma$ to*

$$SNR(\sum_{t=1}^{T} \Delta w^t) = \frac{\mathbb{E}[\sum_{t=1}^{T} s^t + n^t]}{\sqrt{\mathbb{V}[\sum_{t=1}^{T} s^t + n^t]}} \tag{13}$$

$$\geq \frac{\sum_{t=1}^{T} s^t}{\sqrt{\sigma^2(T(1+\alpha)+1)}} \approx \frac{\sqrt{T}}{\sqrt{1+\alpha}} \frac{\bar{s}}{\sigma} \tag{14}$$

with $\bar{s} = \frac{1}{T}\sum_{t=1}^{T} s^t$ being the signal-average over time.

This means that a weight-update will be more informative the longer the accumulation period and the stronger the noise correlates temporally. Convergence speed will not be compromised for as long as the information content of the accumulated update is equal to the cumulative information content of the individual updates (c.f. Fig. 2 (c)). This line of reasoning helps to shed light on both the successes of

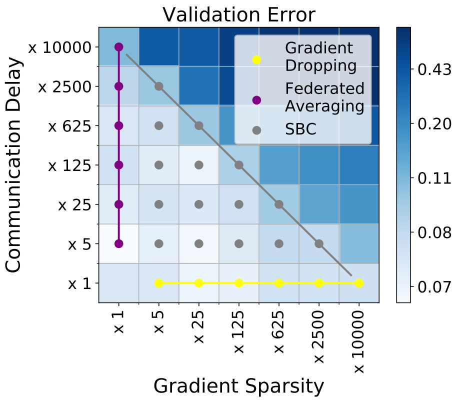

Fig. 3: Validation Error for ResNet32 trained on CIFAR at different levels of temporal and gradient sparsity (the error is color-coded, brighter means lower error). The prior approaches of Gradient Dropping and Federated Averaging can be embedded in a two-dimensional compression framework.

communication delay and gradient sparsification. In fact, it implies that both of these approaches are actually very similar in the way they affect the information flow from client to server on the individual weight level.

We find that this intuition is also verified empirically. Figure 3 shows validation errors for ResNet32 model trained on CIFAR for 60000 iterations at different levels of communication delay and gradient sparsity. We observe multiple things: 1.) The validation error remains more or less constant along the off-diagonals of the matrix where the total sparsity (i.e. the product of communication delay and gradient sparsity) is constant. 2.) The existing methods of Federated Averaging [11] (purple) and Gradient Dropping/ DGC [14] [15](yellow) are just lines in the two-dimensional space of possible compression methods. 3.) There exists a roughly triangular area of approximately constant error, *optimal compression methods lie along the hypotenuse of this triangle*. These results indicate, that communication delay and sparsification with error accumulation affect the convergence in a roughly multiplicative way and that there seems to exist a fixed information budged in DSGD, necessary to maintain unhindered convergence.

In the following we present a framework that allows us to smoothly trade of these two types of gradient accumulation against one another. By doing so our proposed framework can adapt to the requirements of the distributed learning environment and achieve state-of-the-art compression results by reaping the benefits from both approaches.

III. Sparse Binary Compression

Inspired by our findings in the previous section, we propose Sparse Binary Compression (c.f. Fig. 4), to drastically reduce the number of communicated bits in distributed training. SBC makes use of multiple compression techniques *simultaneously*[1] to reduce all multiplicative components of (1).

In the following $\mathcal{W}$ will refer to the entirety of neural network parameters, while $W \in \mathcal{W}$ will refer to one specific tensor of weights. Arithmetic operations on $\mathcal{W}$ are to be understood componentwise.

Algorithm 1: Synchronous Distributed Stochastic Gradient Descent (DSGD)

```
input: initial parameters $\mathcal{W}$
outout: improved parameters $\mathcal{W}$
init: all clients $C_i$ are initialized with the same
  parameters $\mathcal{W}_i \leftarrow \mathcal{W}$, the initial global weight update
  and the residuals are set to zero $\Delta\mathcal{W}, \mathcal{R}_i \leftarrow 0$
for $t = 1, .., T$ do
    for $i \in I_t \subseteq \{1, .., M\}$ in parallel do
        Client $C_i$ does:
        • msg $\leftarrow$ download$_{S \rightarrow C_i}$(msg)
        • $\Delta\mathcal{W} \leftarrow$ Golomb_decode(msg)

        • $\mathcal{W}_i \leftarrow \mathcal{W}_i + \Delta\mathcal{W}$
        • $\Delta\mathcal{W}_i \leftarrow \mathcal{R}_i + \text{SGD}_n(\mathcal{W}_i, D_i) - \mathcal{W}_i$
        • $\Delta\mathcal{W}_i^* \leftarrow \text{STC}(\Delta\mathcal{W}_i)$
        • $\mathcal{R}_i \leftarrow \Delta\mathcal{W}_i - \Delta\mathcal{W}_i^*$

        • msg$_i \leftarrow$ Golomb_encode($\Delta\mathcal{W}_i^*$)
        • upload$_{C_i \rightarrow S}$(msg$_i$)
    end
    Server $S$ does:
    • gather$_{C_i \rightarrow S}(\Delta\mathcal{W}_i^*), \ i \in I_t$
    • $\Delta\mathcal{W} \leftarrow \frac{1}{|I_t|} \sum_{i \in I_t} \Delta\mathcal{W}_i^*$
    • $\mathcal{W} \leftarrow \mathcal{W} + \Delta\mathcal{W}$
    • broadcast$_{S \rightarrow C_i}(\Delta\mathcal{W}), \ i = 1, .., M$
end
return $\mathcal{W}$
```

Algorithm 2: Sparse Binary Compression

```
input: tensor $\Delta W$, sparsity $p$
output: sparse tensor $\Delta W^*$
• val$^+ \leftarrow$ top$_{p\%}(\Delta W)$; val$^- \leftarrow$ top$_{p\%}(-\Delta W)$
• $\mu^+ \leftarrow$ mean(val$^+$); $\mu^- \leftarrow$ mean(val$^-$)
if $\mu^+ \geq \mu^-$ then
    return $\Delta W^* \leftarrow \mu^+(W \geq \min(\text{val}^+))$
else
    return $\Delta W^* \leftarrow -\mu^-(W \leq -\min(\text{val}^-))$
end
```

Communication Delay, Fig. 4 (a): We use communication delay, proposed by [11], to introduce temporal sparsity into DSGD. Instead of communicating gradients after every local iteration, we allow the clients to compute more informative updates by performing multiple iterations of SGD. These generalized weight updates are given by

$$\Delta\mathcal{W}_i = \text{SGD}_n(\mathcal{W}_i, D_i) - \mathcal{W}_i$$

[1] To clarify, we have put our contributions in emphasis.

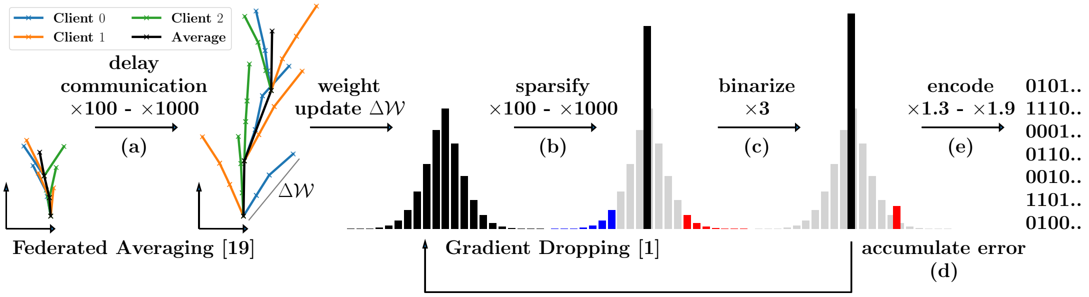

Fig. 4: Step-by-step explanation of techniques used in Sparse Binary Compression: (a) Illustrated is the traversal of the parameter space with regular DSGD (left) and Federated Averaging (right). With this form of communication delay, a bigger region of the loss surface can be traversed, in the same number of communication rounds. That way compression gains of up to ×1000 are possible. After a number of iterations, the clients communicate their locally computed weight updates. (b) Before communication, the weight update is first sparsified, by dropping all but the fraction p weight updates with the highest magnitude. This achieves up to ×1000 compression gain. (c) Then the sparse weight update is binarized for an additional compression gain of approximately ×3. (d) Finally, we optimally encode the positions of the non-zero elements, using Golomb encoding. This reduces the bit size of the compressed weight update by up to another ×2 compared to naive encoding.

Algorithm 3: Golomb Position Encoding

```
input: sparse tensor $\Delta W^*$, sparsity $p$
output: binary message msg
• $\mathcal{I} \leftarrow \Delta W^*[:]_{\neq 0}$
• $\mathrm{b}^* \leftarrow 1 + \lfloor \log_2(\frac{\log(\phi-1)}{\log(1-p)}) \rfloor$
for $i = 1, .., |\mathcal{I}|$ do
    • $d \leftarrow \mathcal{I}_i - \mathcal{I}_{i-1}$
    • $q \leftarrow (d-1)$ div $2^{\mathrm{b}^*}$
    • $r \leftarrow (d-1)$ mod $2^{\mathrm{b}^*}$
    • msg.add($\underbrace{1, .., 1}_{q \text{ times}}$, 0, binary$_{\mathrm{b}^*}(r)$)
end
return msg
```

where $\mathrm{SGD}_n(\mathcal{W}_i, D_i)$ refers to the set of weights obtained by performing n iterations of stochastic gradient descent on $\mathcal{W}_i$, while sampling mini-batches from the i-th client's training data D_i. Empirical analysis by [11] suggests that communication can be delayed drastically, with only marginal degradation of accuracy. For $n = 1$ we obtain regular DSGD.

Sparse Binarization, Fig. 4 (b), (c): Inspired by the works of [15] [12] [10] and [14] we use the magnitude of an individual weight within a weight update as a heuristic for it's importance. First, we set all but the fraction p biggest and fraction p smallest weight updates to zero. Next, we compute the mean of all remaining positive and all remaining negative weight updates independently. *If the positive mean μ^+ is bigger than the absolute negative mean μ^-, we set all negative values to zero and all positive values to the positive mean and vice versa*. The method is illustrated in Fig. 4 and formalized in Algorithm 2. Finding the fraction p smallest and biggest values in a vector W requires $\mathcal{O}(|W|)$ operations, where $|W|$ refers to the number of elements in W [26]. [15] suggest to reduce the computational cost of this operation, by randomly subsampling from W. However this comes at the cost of introducing (unbiased) noise in the amount of sparsity. Luckily, in our approach communication rounds (and thus compressions) are relatively infrequent, which helps to marginalize the overhead of the sparsification. *Quantizing the non-zero elements of the sparsified weight update to the mean reduces the required value bits $\bar{b}_{val}$ from 32 to 0*. This translates to a reduction in communication cost by a factor of around ×3. We can get away with averaging out the non-zero weight updates because they are relatively homogeneous in value and because we accumulate our compression errors.

Residual Accumulation, Fig. 4 (d): It is well established [15] [12] [14] [20] that the convergence in sparsified DSGD can be greatly accelerated by accumulating the error that arises from only sending sparse approximations of the weight updates. In Section II we showed that under simplifying assumptions on the SGD noise distribution residual accumulation reduces the SNR of the weight-updates, thus making them more informative. After every communication round, the residual is updated via

$$\mathcal{R}_\tau = \sum_{t=1}^{\tau}(\Delta\mathcal{W}_t - \Delta\mathcal{W}_t^*) = \mathcal{R}_{\tau-1} + \Delta\mathcal{W}_\tau - \Delta\mathcal{W}_\tau^*. \quad (15)$$

Optimal Position Encoding, Fig. 4 (e): To communicate a set of sparse binary tensors produced by SGC, we only need to transfer the positions of the non-zero elements in the flattened tensors, along with one mean value (μ^+ or μ^-) per tensor. Instead of communicating the absolute non-zero positions it is favorable to only communicate the distances between all non-zero elements. It is possible to show that for big values of $|W|$ and $k = p|W|$, the distances are approximately geometrically distributed with success probability equal to the sparsity rate p. Therefore, we can optimally encode the distances using the Golomb code [27]. Golomb encoding reduces the average number of position bits to

$$\bar{\mathrm{b}}_{pos} = \mathbf{b}^* + \frac{1}{1-(1-p)^{2^{\mathbf{b}^*}}}, \tag{16}$$

with $\mathbf{b}^* = 1 + \lfloor \log_2(\frac{\log(\phi-1)}{\log(1-p)}) \rfloor$ and $\phi = \frac{\sqrt{5}+1}{2}$ being the golden ratio. For a sparsity rate of i.e. $p = 0.01$, we get $\bar{\mathrm{b}}_{pos} = 8.38$, which translates to ×1.9 compression, compared to a naive distance encoding with 16 fixed bits. *While the overhead for encoding and decoding makes it unproductive to use Golomb encoding in the situation of [12], this overhead becomes negligible in our situation due to the infrequency of weight update exchange resulting from communication delay.* The encoding scheme is given in Algorithm 3.

Momentum Correction, Warm-up Training and Momentum Masking: [15] introduce multiple minor modifications to the vanilla Gradient Dropping method, to improve the convergence speed. We adopt momentum masking, while momentum correction is implicit to our approach.

Our proposed method is described in Algorithms 1, 2 and 3. Algorithm 1 describes how compression and residual accumulation can be introduced into DSGD. Algorithm 2 describes our compression method. Algorithm 3 describes the Golomb encoding. Table I compares theoretical asymptotic compression rates of different popular compression methods.

TABLE I: Theoretical asymptotic compression rates for different compression methods broken down into components. Only SBC reduces all multiplicative components of the total bitsize (c.f. (1)).

Total Bits =		Baseline	SignSGD, TernGrad, QSGD	DGC, Strom	Federated Averaging	STC
	Delay	100%	100%	100%	**0.1% - 10%**	**0.1% - 10%**
×	Sparsity	100%	100%	**0.1%**	100%	**0.1% - 10%**
$\times \sum$	Value Bits	32	**1 - 8**	32	32	**1**
	Pos. Bits	0	0	16	0	**8 - 14**
Compression Rate		×**1**	×**4** - ×**32**	×**666**	×**10** - ×**1000**	- ×**40000**

IV. Experiments

A. Networks and Datasets

We evaluate our method on commonly used convolutional and recurrent neural networks with millions of parameters, which we train on well-studied data sets that contain up to multiple millions of samples. We perform experiments with client numbers ranging from 4 to 400 to cover both the distributed training and federated learning use-case.

Image Classification: We run experiments for LeNet5-Caffe[2] on MNIST [29], ResNet18 and ResNet34 [4] on CIFAR-10 and CIFAR-100 [30] and ResNet50 on ILSVRC12 (ImageNet) [31]. We split the training data randomly into equally sized shards and assign one shard to every one of the clients. All models are trained using momentum SGD, except for LeNet5-Caffe, which is trained using the Adam optimizer [32]. Learning rate, weight intitiallization and data augmentation are as in the respective papers.

TABLE II: Final accuracy/perplexity achieved on the test split and average compression rate for different compression schemes in a distributed training setting with four clients on different models and datasets.

Compression Method ⟶		Baseline	DGC [3]	Fed. Avg.[4]	SBC (1)	SBC (2)	SBC (3)
LeNet5-Caffe @MNIST	Accuracy	0.9946	0.994	0.994	0.994	0.994	0.991
	Compression	×1	×718	×500	×2071	×3166	×24935
ResNet18 @CIFAR10	Accuracy	0.946	0.9383	0.9279	0.9422	0.9435	0.9219
	Compression	×1	×768	×1000	×2369	×3491	× 31664
ResNet34 @CIFAR100	Accuracy	0.773	0.767	0.7316	0.767	0.7655	0.701
	Compression	×1	×718	×1000	×2370	×3166	×31664
ResNet50 @ImageNet	Accuracy	0.737	0.739	0.724	0.735	0.737	0.728
	Compression	×1	×601	×1000	×2569	×3531	×37208
WordLSTM @PTB	Perplexity	76.02	75.98	76.37	77.73	78.19	77.57
	Compression	×1	×719	×1000	×2371	×3165	×31658
WordLSTM* @WIKI	Perplexity	101.5	102.318	131.51	103.95	103.95	104.62
	Compression	×1	×719	×1000	×2371	×3165	×31657

Language Modeling: We experiment with multilayer sequence-to-sequence LSTM models as described in [33] on the Penn Treebank (PTB) [34] and Wikitext-2 corpora for next-word prediction. The PTB dataset consists of a sequence 923000 training, and 82000 validation words, while the Wikitext-2 dataset contains 2088628 train and 245569 test words. On both datasets we train a two-layer LSTM model with 650 and 200 hidden units respectively ("WordLSTM" / "WordLSTM*") with tied weights between encoder and decoder as described in [35]. The training data is split into consecutive subsequences of equal length, out of which we assign one to every client.

While the models we use in our experiments do not fully achieve state-of-the-art results on the respective tasks and datasets, they are still sufficient for the purpose of evaluating our compression method and demonstrate, that our method works well with common regularization techniques such as batch normalization [36] and dropout [37].

B. Results

We experiment with three configurations of our method: SBC (1) uses no communication delay and a gradient sparsity of 0.1%, SBC (2) uses 10 iterations of communication delay and 1% gradient sparsity and SBC (3) uses 100 iterations of communication delay and 1% gradient sparsity. Our decision for these points on the 2D grid of possible configurations is somewhat arbitrary. The experiments with SBC (1) serve the purpose of enabling us to directly compare our 0-value-bit quantization to the 32-value-bit Deep Gradient Compression [15]).

Table II lists compression rates and final validation accuracies achieved by different compression methods, when applied to the training of neural networks on 5 different datasets. The number of iterations (forward-backward-passes) is held constant for all methods. On all benchmarks, our methods perform comparable to the baseline in terms of achieved accuracy, while communicating significantly less bits.

[2] A modified version of LeNet5 from [28].

[3] At a sparsity rate $p = 0.1\%$ without warm-up training. The gradients are encoded with Golomb encoding prior to communication.

[4] The compression rate is equal to the communication delay.

TABLE III: Final accuracy achieved on the test split and average compression rate for different compression schemes in a federated learning setting with larger numbers of clients.

	Compression Method ⟶		Baseline	Gradient Droping	Federated Averaging	SBC (1)	SBC (2)
50	ResNet18*[5] @CIFAR10	Accuracy	0.9254	0.9167	0.911	0.921	0.902
		Compression	×1	×713	×100	×2362	×3166
100	LeNet5-Caffe @MNIST	Accuracy	0.979	0.9811	0.967	0.979	0.9818
		Compression	×1	×714	×100	×2363	×3165
400	LeNet5-Caffe @MNIST	Accuracy	0.9758	0.9744	0.899	0.9731	0.9733
		Compression	×1	×714	×100	×2363	×3165

Figure 6 shows convergence speed in terms of iterations for ResNet50 trained on ImageNet. The convergence speed is only marginally affected, by SBC. In the first 30 epochs SBC (3) even achieves the highest accuracy, using about ×37000 less bits than the baseline. In total, SBC (3) reduces the upstream communication on this benchmark from 125 *terabytes* to 3.35 gigabytes for every participating client (c.f. Fig. 5). After the learning rate is lowered in epochs 30 and 60 progress slows down for SBC (3) relative to the methods which do not use communication delay. In direct comparison SBC (1) performs very similar to Gradient Dropping, while using about ×4 less bits (that is ×2569 less bits than the baseline).

Next, we investigate the behavior of our method for higher numbers of clients. Table III shows results for a training setting with 50, 100 and 400 clients on MNIST and CIFAR10. We can see that as the number of clients grows, Federated Averaging starts to noticeably slow down the convergence and degrades the final accuracy, even if we reduce the delay period to only 100 iterations. Our methods SBC (1) and (2) that rely more heavily on gradient sparsification perform much better in this setting and in some cases and closely match the baseline. This behavior is expected, as the more frequent exchange of gradient information in SBC (1) and (2) keeps all clients aligned, while they diverge further from one another for every iteration that communication is delayed in Federated Averaging. For detailed evaluation of federated learning in the non-i.i.d. setting, we refer the reader to [38].

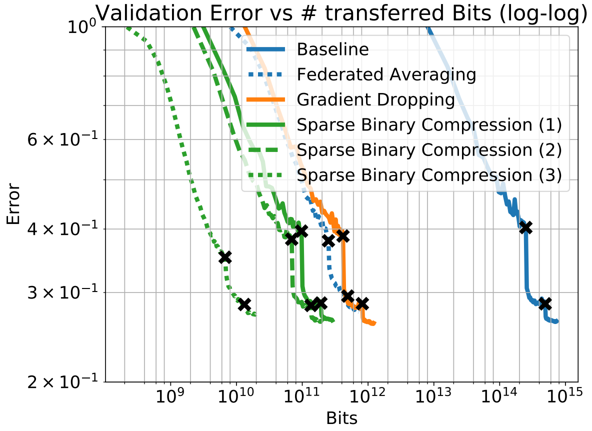

Fig. 5: Top-1 validation error vs number of transferred bits (log-log) for ResNet50 trained on ImageNet using different methods for compressed communication. Epochs 30 and 60 at which the learning rate is reduced are marked in the plot. Sparse Binary Compression converges similarly fast as the uncompressed baseline in terms of SGD iterations, while communicating *four orders of magnitude* less bits.

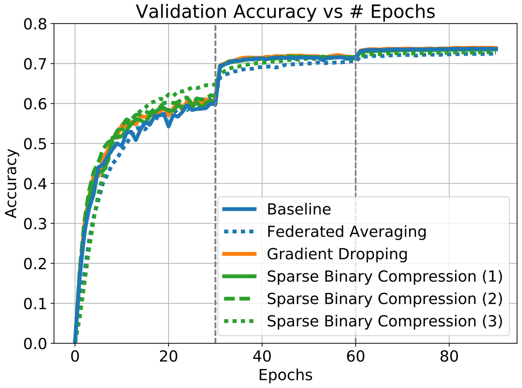

Fig. 6: Convergence speed of different methods for compressed communication for ResNet50 trained on ImageNet. Epochs 30 and 60 at which the learning rate is reduced are marked in the plot.

V. Conclusion

The gradient information for training deep neural networks with SGD is highly redundant (see e.g. [15]). We exploit this fact to the extreme by combining 3 powerful compression strategies and are able to achieve compression gains of up to *four orders* of magnitude with only a slight decrease in accuracy. More fundamentally, we present theoretical and empirical evidence suggesting that the formerly treated as separate compression methods of communication delay and gradient sparsification in fact can be viewed as two very similar forms of gradient delay that affect the convergence speed in a roughly multiplicative way. Based on this insight we propose a framework that is able to reap the benefits from both compression approaches and can smoothly adapt to communication-constraints in the learning environment, such as network bandwidth and latency, and (SGD-)computation time as well as temporal inhomogeneities therein. This leads to advantages in both federated learning and data-parallel training of deep neural networks.

It remains an interesting direction of further research to perform federated learning with compressed representations of deep neural networks (e.g. [39]) and to investigate the learning process and the changes in representation with explanation methods [40], [41]. Furthermore, we aim to identify heuristics and theoretical insights that can help to choose the optimal balance between communication delay and sparsity to guide compression further towards optimality.

[5] ResNet18* only has half as many convolutional filters in every layer as ResNet18.

Acknowledgement

This work was supported by the Fraunhofer Society through the MPI-FhG collaboration project "Theory & Practice for Reduced Learning Machines". This research was also supported by the German Ministry for Education and Research as Berlin Big Data Centre (01IS14013A) and Berlin Center for Machine Learning (01IS18037I). Partial funding by DFG is acknowledged (EXC 2046/1, project-ID: 390685689). This work was also supported by the Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (No. 2017-0-00451).

References

[1] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, A. Senior, P. Tucker, K. Yang, Q. V. Le, *et al.*, "Large scale distributed deep networks," in *Advances in Neural Information Processing Systems (NIPS)*, pp. 1223–1231, 2012.

[2] B. Recht, C. Re, S. Wright, and F. Niu, "Hogwild: A lock-free approach to parallelizing stochastic gradient descent," in *Advances in Neural Information Processing Systems (NIPS)*, pp. 693–701, 2011.

[3] P. Moritz, R. Nishihara, I. Stoica, and M. I. Jordan, "Sparknet: Training deep networks in spark," *arXiv preprint arXiv:1511.06051*, 2015.

[4] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

[5] G. Huang, Z. Liu, K. Q. Weinberger, and L. van der Maaten, "Densely connected convolutional networks," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 1, p. 3, 2017.

[6] T. M. Chilimbi, Y. Suzue, J. Apacible, and K. Kalyanaraman, "Project adam: Building an efficient and scalable deep learning training system.," in *OSDI*, vol. 14, pp. 571–582, 2014.

[7] M. Zinkevich, M. Weimer, L. Li, and A. J. Smola, "Parallelized stochastic gradient descent," in *Advances in Neural Information Processing Systems (NIPS)*, pp. 2595–2603, 2010.

[8] E. P. Xing, Q. Ho, W. Dai, J. K. Kim, J. Wei, S. Lee, X. Zheng, P. Xie, A. Kumar, and Y. Yu, "Petuum: A new platform for distributed machine learning on big data," *IEEE Transactions on Big Data*, vol. 1, no. 2, pp. 49–67, 2015.

[9] M. Li, D. G. Andersen, A. J. Smola, and K. Yu, "Communication efficient distributed machine learning with the parameter server," in *Advances in Neural Information Processing Systems (NIPS)*, pp. 19–27, 2014.

[10] R. Shokri and V. Shmatikov, "Privacy-preserving deep learning," in *22nd ACM SIGSAC Conference on Computer and Communications Security*, pp. 1310–1321, 2015.

[11] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, *et al.*, "Communication-efficient learning of deep networks from decentralized data," *arXiv preprint arXiv:1602.05629*, 2016.

[12] N. Strom, "Scalable distributed dnn training using commodity gpu cloud computing," in *16th Annual Conference of the International Speech Communication Association*, 2015.

[13] Y. Tsuzuku, H. Imachi, and T. Akiba, "Variance-based gradient compression for efficient distributed deep learning," *arXiv preprint arXiv:1802.06058*, 2018.

[14] A. F. Aji and K. Heafield, "Sparse communication for distributed gradient descent," *arXiv preprint arXiv:1704.05021*, 2017.

[15] Y. Lin, S. Han, H. Mao, Y. Wang, and W. J. Dally, "Deep gradient compression: Reducing the communication bandwidth for distributed training," *arXiv preprint arXiv:1712.01887*, 2017.

[16] S. U. Stich, J.-B. Cordonnier, and M. Jaggi, "Sparsified sgd with memory," in *Advances in Neural Information Processing Systems*, pp. 4452–4463, 2018.

[17] D. Alistarh, T. Hoefler, M. Johansson, N. Konstantinov, S. Khirirat, and C. Renggli, "The convergence of sparsified gradient methods," in *Advances in Neural Information Processing Systems*, pp. 5977–5987, 2018.

[18] J. Konečnỳ, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," *arXiv preprint arXiv:1610.05492*, 2016.

[19] W. Wen, C. Xu, F. Yan, C. Wu, Y. Wang, Y. Chen, and H. Li, "Terngrad: Ternary gradients to reduce communication in distributed deep learning," *arXiv preprint arXiv:1705.07878*, 2017.

[20] F. Seide, H. Fu, J. Droppo, G. Li, and D. Yu, "1-bit stochastic gradient descent and its application to data-parallel distributed training of speech dnns," in *15th Annual Conference of the International Speech Communication Association*, 2014.

[21] J. Bernstein, Y.-X. Wang, K. Azizzadenesheli, and A. Anandkumar, "signsgd: compressed optimisation for non-convex problems," *arXiv preprint arXiv:1802.04434*, 2018.

[22] D. Alistarh, D. Grubic, J. Li, R. Tomioka, and M. Vojnovic, "Qsgd: Communication-efficient sgd via gradient quantization and encoding," in *Advances in Neural Information Processing Systems (NIPS)*, pp. 1707–1718, 2017.

[23] S. P. Karimireddy, Q. Rebjock, S. U. Stich, and M. Jaggi, "Error feedback fixes signsgd and other gradient compression schemes," *arXiv preprint arXiv:1901.09847*, 2019.

[24] J. Bernstein, J. Zhao, K. Azizzadenesheli, and A. Anandkumar, "signsgd with majority vote is communication efficient and byzantine fault tolerant," *arXiv preprint arXiv:1810.05291*, 2018.

[25] Y. A. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, "Efficient backprop," in *Neural networks: Tricks of the trade*, pp. 9–48, Springer, 2012.

[26] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*. MIT press, 2009.

[27] S. Golomb, "Run-length encodings (corresp.)," *IEEE Transactions on Information Theory*, vol. 12, no. 3, pp. 399–401, 1966.

[28] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[29] Y. LeCun, "The mnist database of handwritten digits," *http://yann. lecun. com/exdb/mnist/*, 1998.

[30] A. Krizhevsky, V. Nair, and G. Hinton, "The cifar-10 dataset," *online: http://www. cs. toronto. edu/kriz/cifar. html*, 2014.

[31] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 248–255, 2009.

[32] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[33] W. Zaremba, I. Sutskever, and O. Vinyals, "Recurrent neural network regularization," *arXiv preprint arXiv:1409.2329*, 2014.

[34] M. P. Marcus, M. A. Marcinkiewicz, and B. Santorini, "Building a large annotated corpus of english: The penn treebank," *Computational Linguistics*, vol. 19, no. 2, pp. 313–330, 1993.

[35] H. Inan, K. Khosravi, and R. Socher, "Tying word vectors and word classifiers: A loss framework for language modeling," *arXiv preprint arXiv:1611.01462*, 2016.

[36] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International Conference on Machine Learning (ICML)*, pp. 448–456, 2015.

[37] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[38] F. Sattler, S. Wiedemann, K.-R. Müller, and W. Samek, "Robust and communication-efficient federated learning from non-iid data," *arXiv preprint arXiv:1903.02891*, 2019.

[39] S. Wiedemann, K.-R. Müller, and W. Samek, "Compact and computationally efficient representation of deep neural networks," *arXiv preprint arXiv:1805.10692*, 2018.

[40] W. Samek, T. Wiegand, and K.-R. Müller, "Explainable artificial intelligence: Understanding, visualizing and interpreting deep learning models," *ITU Journal: ICT Discoveries - Special Issue 1 - The Impact of Artificial Intelligence (AI) on Communication Networks and Services*, vol. 1, no. 1, pp. 39–48, 2018.

[41] S. Lapuschkin, S. Wäldchen, A. Binder, G. Montavon, W. Samek, and K.-R. Müller, "Unmasking clever hans predictors and assessing what machines really learn," *Nature Communications*, vol. 10, p. 1096, 2019.